# SOFTWARE REQUIREMENT ENGIN EERING (SDC-405)

## Topic number 1: Requirement Engineering

### Definition:

Requirement Engineering is a process used to understand the needs of stakeholders, document them, and manage them in the best possible way.

### Main Features of Requirement Engineering:

#### 1. Elicitation

- This step means gathering requirements from stakeholders.
- It involves asking questions, interviews, or surveys.
- We try to understand what users really need.
- Clear communication is very important in this step.
- Without proper elicitation, wrong features may be built.

#### 2. Analysis

- In this step, we study and understand all gathered requirements.
- We check if the requirements are complete and correct.
- Any conflicts or confusion are solved here.
- This helps in understanding how the system should work.
- Proper analysis avoids problems in the later stages.

## 3. Specification

- Specification means writing the requirements in detail.
- The goal is to make everything clear and easy to follow.
- It is usually written in a document format.
- This document is used by developers and testers.
- It helps everyone understand what to build.

## 4. Verification

- Verification checks if the requirements are written correctly.
- We make sure that nothing important is missing.
- It also ensures the requirements are clear and not confusing.
- Review meetings or checklists are used for this.
- It helps in avoiding mistakes before development starts.

## 5. Validation

- Validation checks if the requirements match stakeholder needs.
- It asks: "Are we building the right system?"
- Feedback from users or clients is taken here.
- If requirements are wrong, we fix them at this stage.
- Validation ensures user satisfaction in the end.

## 6. Management

- This step deals with handling requirement changes.
- Sometimes, users change their needs during the project.
- We update, track, and control all changes properly.
- Good management keeps everything organized and up to date.
- It helps the project stay on the right path.

## Advantages of Requirement Engineering:

- It helps in clearly understanding stakeholder needs.
- It reduces errors by finding problems early.
- It provides clear and structured documentation.
- It makes handling changes easy and organized.
- It saves time and cost by avoiding rework.

# Topic number 2: Software Requirement

## Definition:

It defines that a software must do this and should perform the following work under given constraints.

## Classifications of Software Requirement:

### 1. Functional Requirement

- These define what the software should do.
- They describe specific features and functionalities.

- Example: Login, registration, data saving, etc.
- They are directly related to user needs.
- Functional requirements are essential for system operation.

## 2. Non-Functional Requirement

- These define how the software should perform.
- Focus is on quality aspects like speed, security, or usability.
- Example: The system should load within 3 seconds.
- They improve user experience and system efficiency.
- Non-functional requirements support functional ones.

## 3. Domain Requirement

- These are specific to the application's field or industry.
- They depend on the business environment or rules.
- Example: In banking, transactions must follow financial laws.
- They often come from stakeholders or domain experts.
- Domain requirements ensure the software fits its context.

## NOTE: Library Management System
## Write down its functional and non functional requirements?

### Functional Requirements

These describe what the system should do:

- User Registration and Login
- Book Management
- Book Search
- Issue and Return Books
- Fine Calculation
- User Role Management
- View Borrowing History
- Notification System

## Non-Functional Requirements

These describe **how** the system should behave:

- Performance
- Security
- Scalability
- Usability
- Reliability
- Maintainability
- Backup and Recovery

## Topic : Software Process(Not Include In Outline But Important)

## Definition of Process:

A process is a set of organized activities that work together to change inputs in to useful outputs.

## Definition of Software Process:

A software process is a systematic and organized set of activities used to deve lop, test, and manage softw are applic ations. It focuses on maintaining quality through accuracy, reusability, and sc alability.

## Definition of Software Process Model:

A software process model is the simplest and clear description of how the soft ware development process will be c arried out, step by step.

## Types of Software Process Model

- Waterfall model
- Agile Model
- Scrum Model
- Spiral Model
- DevOps Model

## Waterfall Model

The Waterfall Model is a linear and sequential softw are development process. I t means each phase is completed one by one in a specific order — like a waterf all, where w ater flows down step by step.

### Key Features:

- Linear and Sequential Process
- Used for Small and Simple Projects

- Less Communication with Client
- Less Satisfaction of Client
- Have Prediction

## Waterfall Model Phases:

### 1. Requirement

Collecting complete requirements from the client.

### 2. Analysis

Understanding and analyzing the requirements.

### 3. Design

Creating system and software design based on requirements.

### 4. Development

Actual coding and converting design into a working system.

### 5. Testing

Checking for bugs and ensuring the software works correctly.

### 6. Implementation

Delivering the software to the client for use.

### 7. Maintenance

Fixing issues or making improvements after the software is in use.

## Agile Model

The Agile Model is a flexible and iterative software development process. It me ans the work is done in small parts (called iterations or sprints), and feedback i

s taken regularly from the client to make improvements.

## Key Features:

- Flexible and Iterative Process
- Best for Medium to Large and Complex Projects
- Frequent Communication with Client
- High Client Satisfaction
- Less Prediction, More Adaptation

## Agile Model Phases:

- **Requirement**
  Only basic requirements are gathered at the start. More details are added in each sprint.

- **Planning**
  Plan a short development cycle (sprint) and decide which features to work on.

- **Design**
  Make a simple and flexible design that can change based on feedback.

- **Development**
  Develop the selected features in each sprint and deliver working software.

- **Testing**
  Testing is done in every sprint to find and fix issues early.

- **Review**
  Client gives feedback after every sprint, and changes are made accordingly.

- **Delivery**
  After several sprints, the complete product is delivered with all updates and feedback included.

# Scrum Model

The Scrum Model is an agile-based software development process. It works in short cycles called **sprints** where teams deliver small parts of the software and take feedback regularly to improve the product.

## Key Features:

- Work Done in Short Cycles (Sprints)
- Used for Medium to Large Projects
- Daily Communication with Client and Team
- High Client Satisfaction
- Flexible and Quick Changes Possible

## Scrum Model Phases:

- **Requirement**
  Basic features are written as *user stories* in a list called the **Product Back log**.

- **Planning**
  The team selects a few user stories from the backlog to complete in a sp rint (usually 1–4 weeks).

- **Design**
  Simple and easy design is created based on selected features.

- **Development**
  Coding starts for the selected features in the sprint.

- **Testing**
  Each feature is tested during the sprint to make sure it works properly.

- **Review**
  At the end of each sprint, the team shows the work to the client for feedb ack.

- **Retrospective & Maintenance**
  Team discusses what went well and what can be improved. Fixes and updates continue in the next sprints.

## DevOps Model

The DevOps Model is a modern software development and operations process. It combines **Development (Dev)** and **Operations (Ops)** teams to work together for faster and better delivery of software.

### Key Features:

- Development and Operations Work Together
- Used for Fast and Continuous Delivery
- More Automation and Monitoring
- Strong Communication Between Teams
- Faster Updates and Fixes

### DevOps Model Phases:

- **Requirement**
  Gather client needs and plan features that are valuable and can be delivered quickly.

- **Planning & Development**
  Teams plan and write code using collaboration tools. Development is continuous.

- **Build & Testing**
  Code is automatically built and tested using tools. Bugs are fixed early.

- **Release**
  Tested code is prepared and released for use quickly and safely.

- **Deploy**
  Software is deployed (launched) automatically to live systems using automation tools.

- **Operate**
  System is monitored continuously to make sure it runs smoothly.

- **Monitor & Feedback**
  Performance is tracked and feedback is collected for improvements in the next cycle.

## Spiral Model

The Spiral Model is a risk-driven software development process. It combines features of both Waterfall and iterative models. Work is done in repeated cycles (spirals), and each cycle includes planning, risk checking, development, and testing.

### Key Features:

• Combination of Waterfall and Iterative Model
• Used for Large and Risky Projects
• Frequent Risk Analysis
• Client Involvement in Every Phase
• Flexible and Repeatable Process

### Spiral Model Phases:

- **Requirement**
  Basic requirements are collected in the first spiral and updated in later spirals.

- **Risk Analysis**
  Possible risks are identified and plans are made to avoid or solve them.

- **Design**
  A rough design is created, then improved in each spiral based on feedback.

- **Development**
  The system is developed in parts. Each spiral adds more functionality.

- **Testing**
  Every part is tested separately to make sure it works before moving forward.

- **Implementation**
  After several spirals, the complete software is delivered to the client.

- **Maintenance**
  Updates, bug fixes, and changes are made regularly after delivery.

## Topic number 3: Levels/Layers of Requirement

This is same as the main features of requirement engineering.

## Topic number 4: Requirement Characteristics

### Definition:

**Requirement Characteristics** refer to the **qualities or features** that a good software requirement should have. These characteristics help ensure that the requirements are **clear, useful, and effective** for building the software correctly.

### Main Characteristics of Good Requirements:

#### 1. Correct

The requirement should be exactly what the user wants.
It should not have any mistakes.
It must reflect the real needs of the system.

#### 2. Complete

The requirement should have all the needed details.
Nothing important should be missing.
It should cover every part of the functionality.

### 3. Clear and Unambiguous

The requirement should be written in simple and clear words.
It should not confuse the reader.
There must be only one meaning.

### 4. Consistent

All requirements should agree with each other.
There should be no conflicts between them.
They must not give opposite meanings.

### 5. Verifiable

You should be able to test or check the requirement.
It must be measurable or reviewable.
If it can't be tested, it's not verifiable.

### 6. Feasible

The requirement should be possible to do with given time and tools.
It should not demand things that can't be done.
It must be practical to implement.

### 7. Necessary

Only needed features should be included.
Extra or unimportant requirements should be avoided.
Each requirement must have a reason to be there.

### 8. Traceable

Every requirement should link back to its source.
You must know where it came from (like user or business goal).
This helps in tracking changes and purpose.

## Topic number 5: Requirement Evolution

## Definition

**Requirement evolution** is the process of continuous change and refinement of software requirements throughout the software development life cycle — from elicitation to maintenance.

## Reasons for Requirement Evolution:

### 1. Stakeholder Changing Requirements

Stakeholders may change their needs or expectations during the project.
This can happen due to market trends, user feedback, or business goals.
So, the development team must update the requirements.

### 2. Change in Tools and Technology

New technologies or tools may become available during development.
These tools might offer better performance or new features.
So, requirements evolve to take advantage of them.

### 3. Wrong Assumptions in the Beginning

Sometimes, developers or clients make early assumptions that turn out to be w rong.
When these are found later, the requirements need to be corrected.
This ensures the software works properly in the real environment.

### 4. Legal or Regulatory Changes

Laws or industry standards may change over time.
To stay compliant, the software must meet the new rules.
So, requirements are updated according to legal needs.

### 5. Competitive Pressure

Competitors may release better or faster software.
To stay in the market, clients may ask for new or improved features.

This leads to changes in existing requirements.

## Types of Requirement Evolution:

## 1. Automation Evolution

### Definition:

Automation evolution refers to the process of updating requirements when ma nual tasks are replaced by automated systems.

### Explanation:

Automation evolution happens when companies want to reduce manual work.
They update the system to do tasks automatically, like using machines or soft ware tools.
This saves time and lowers the chance of human mistakes.
For example, replacing manual data entry with barcode scanning is automatio n.
So, requirements are changed to add support for these new automatic feature s.

## 2. Technology Evolution

### Definition:

Technology evolution means the change or upgrade in tools, platforms, or tech nologies used during software development, which causes changes in require ments.

### Explanation:

New tools and technologies keep coming in the IT world.
When better technology is available, old systems may become outdated.
To keep the softw are fast, secure, and modern, the requirements must be chan ged.
For example, moving from desktop to mobile apps needs new planning and fea

tures.
So, technology changes lead to updated software requirements.

# Topic number 6: Requirement Analysis

## Definition:

Requirement analysis is the systematic examination of requirements to ensure they are correct, well-defined, and aligned with stakeholders' needs and project goals.

### Important Impact of Requirement Analysis:

1. Validation

**Definition:** Validation checks whether the final product meets the user's actual needs.

Key Points:

- It ensures that the right product is being built for the stakeholders.
- Helps in identifying missing or misunderstood requirements early.
- It reduces the risk of user dissatisfaction after development.

2. Verification

**Definition:** Verification ensures that the requirements are written correctly and clearly.

Key Points:

- It checks that the requirements are complete, consistent, and clear.
- Makes sure the document follows proper format and standard.
- Helps developers understand exactly what to build.

### 3. Elicitation

**Definition:** Elicitation is the process of gathering requirements from stakeholders.

Key Points:

- Involves techniques like interviews, surveys, and meetings.
- Helps in understanding what users expect from the system.
- It forms the foundation for accurate requirement documentation.

### 4. Traceability

**Definition:** Traceability tracks each requirement through all stages of development.

Key Points:

- Helps link requirements to their design, code, and test cases.
- Makes it easy to check which requirements are implemented.
- Supports impact analysis when any requirement changes.

## Techniques of Requirement Analysis:

### 1. Use Case

**Definition:** Use cases describe how users will interact with the system.

Key Points:

- Shows the step-by-step actions a user performs in a system.
- Helps understand functional requirements from the user's view.
- Makes it easier to find missing or extra features early.

## 2. Flow Chart

**Definition:** A flow chart is a diagram that shows the flow of processes in a syst em.

Key Points:

- Visualizes how data or actions move through a system.
- Helps in identifying errors, loops, or inefficiencies.
- Easy to understand for both technical and non-technical people.

## 3. Gap Analysis

**Definition:** Gap analysis finds the difference between the current state and the d esired future state.

Key Points:

- Identifies what is missing to reach project goals.
- Helps plan new features or improvements needed in the system.
- Useful in upgrading or replacing old systems.

## 4. Prototyping

**Definition:** Prototyping creates a simple model or mock-up of the system befor e full development.

Key Points:

- Gives stakeholders a visual idea of the system early.
- Helps collect feedback and make changes quickly.
- Saves time and cost by reducing rework later.

## Advantages of Requirement Analysis:

### Clear Understanding of Needs
– It helps the development team understand exactly what the users and stakeholders want.

### Reduces Project Errors
– Early analysis prevents mistakes like missing or wrong features during development.

### Saves Time and Cost
– By catching problems early, it avoids costly changes in later stages.

### Improves Communication
– It creates better communication between developers, clients, and stakeholders.

### Better Quality Software
– When requirements are clear and accurate, the final product meets expectations and works smoothly.

# Topic number 7: Requirement Traceability

## Definition:

Requirement traceability is the process of tracking and linking requirements throughout the software development life cycle (SDLC), ensuring that each requirement is fulfilled at every stage of development.

## Advantages of Requirement Traceability:

1. **Ensure every requirement is fulfilled and tested:**
   It helps make sure all requirements are completed and properly tested.

2. **You can manage changes easily:**
   Any change in requirements can be tracked and updated without confusion.

3. Improve project transparency:
   Everyone can clearly see the progress and status of each requirement.

4. Reduce mistakes:
   Tracking requirements helps avoid missing or incorrect features.

5. Support better decision-making:
   Clear links between requirements and work done help in making smart pr oject choices.

## Features of Requirement Traceability:

1. Unique ID for each requirement:
   Every requirement is given a special ID to track it easily through the proje ct.

2. Regular reviews:
   The requirements are checked regularly to ensure they are still correct an d complete.

3. Automated traceability tools:
   Tools are used to automatically link and track requirements, saving time and reducing errors.

4. Use of traceability matrix:
   A traceability matrix is used to map and connect requirements with desi gn, development, and testing.

## Types of Requirement Traceability:

## Forward Traceability

Definition:

Bidirectional traceability is the ability to trace both forward (requirements to pr oduct) and backward (product to requirements).

Key Points:

- Gives complete visibility of how each requirement is handled.
- Helps detect any missing or extra functionality in the system.
- Ensures strong control over changes and project scope.

## Backward (or Reverse) Traceability

Definition:

Backward traceability means tracing from the final product or test cases back to the original requirements.

Key Points:

- Helps verify that no extra or unnecessary features are added.
- Useful in identifying the reason or origin behind each part of the system.
- Ensures all system functions are based on actual requirements.

## Bidirectional Traceability

Definition:

Bidirectional traceability is the ability to trace both forward (requirements to product) and backward (product to requirements).

Key Points:

- Gives complete visibility of how each requirement is handled.
- Helps detect any missing or extra functionality in the system.
- Ensures strong control over changes and project scope.

## Vertical Traceability

Definition:

Vertical traceability shows the relationship between requirements and different development levels — like from user requirements to system requirements to code.

Key Points:

- Helps connect high-level business needs with low-level technical work.
- Improves understanding across different roles (managers, developers, testers).
- Useful in validating that the final product meets original goals.

## Horizontal Traceability

Definition:

Horizontal traceability checks the relationship between items within the same level of development — like between design components or test cases.

Key Points:

- Helps identify dependencies among components at the same level.
- Useful for ensuring consistency across similar modules or processes.
- Makes it easier to track how changes in one part affect others on the same level.